Jinja2-TD

Release 3.1.2

Louis DEVIE

Nov 04, 2022

CONTENTS:

Provides information about Jinja2 template dependencies.

The way it's done is quite intrusive, but the idea is that this package does all the dirty work so you don't have to. It provides an interface that lets you use it without having to modifiy existing Jinja code.

Be aware that by importing the package `jinja2td` or one of its modules, you alter the way jinja works, and that existing jinja code may break or get slower.

CHAPTER

ONE

SETUP

1.1 Installation

You can install it with pip :

```
pip install Jinja2-TD==3.x.x
```

The Jinja2-TD version must match your Jinja2 version. Currently, only the latest version (3.1.2) is supported.

1.2 Import and setup

Then, import it in Python:

```
import jinja2td
```

And add the `Introspection` extension to a Jinja environment:

```
env = jinja2.Environment(
   extensions=[Introspection, ...],
   ...
)
```

You can now start using it!

CHAPTER

TWO

BASIC USAGE

Once you've successfully setup `jinja2td`, you can :

2.1 Get information about a template

If a template has been loaded in the environment (e.g. with `get_template`), you can use the `dependencies` attribute of the environment retrieve the static dependencies of that template:

```
...

my_template = env.get_template("my_template.j2")

...

# use the same template name
template_info = env.dependencies.get_template("my_template.j2")

# get the parent
parent = template_info.get_parent()
if parent is not None:
    print("This template extends", parent.target.name)
else:
    print("This template doesn't extend another")

# find wich templates {% include %} this one
print("This template is included in:")
for t in template_info.find_included():
    print(t.name)

...
```

Note: Dynamic dependencies (e.g. `{% extends variable %}`) are detected, but will be ignored by functions like `get_parent` and `find_included`. See the *API reference* for more information about dynamic dependency handling.

2.2 Find the templates actually used

You can detect all the templated used during render with the `watch` and `used_last_watch` functions:

```
...

my_template = env.get_template("my_template.j2")

# call before each render you want to watch
env.dependencies.watch()

result = my_template.render(...)

templates_used = env.depedencies.used_last_watch()
print("Templates used to render my_template.j2:")
for t in templates_used:
    print(t.name, t.file)

...
```

CHAPTER

THREE

API REFERENCE

class jinja2td.**DependencyGraph**

A collection of templates and their dependencies.

This is the type of the `dependencies` attribute of the environment.

get_template(*name: str*) → Optional[*Template*]

Get a template.

Parameters

name – The name of the template, the same you would pass to jinja2.Environment.get_template.

Returns

The corresponding template, or None if the template is unknown.

property templates: List[*Template*]

All the templates known to the environment.

used_last_watch() → List[*Template*]

Returns all the templates used for rendering templates since the last call to *watch*.

By deault, the watch system is disabled in async environments, because it gets messy when multiple jinja2.Template.render_async run in parallel. You can enable it explicitly by setting *watch_async* to `True`, but *be careful not to call watch or used_last_watch while a template is rendering*.

Returns

The names of the templates used during the last watch.

watch()

Start watching for templates used.

Call it before rendering a template, and then use *used_last_watch* to get all the templates used to build it.

property watch_async: bool

See *used_last_watch*.

class jinja2td.**Template**(*name: str*, *file: Optional[str]*, *graph:* DependencyGraph)

Represent a template in the dependency graph.

This is NOT a Jinja2 template.

property dependencies: List[*Dependency*]

The dependencies of this template.

property file: Optional[str]

The source file of the template, or `None` if the template wasn't loaded from a file.

find_children() → List[*Template*]

Get the templates extending this one.

Returns
The list of templates with an `"extends"` dependency targeting this template.

find_imported() → List[*Template*]

Get the templates that import this one.

Returns
The list of templates with an `"import"` dependency targeting this template.

find_included() → List[*Template*]

Get the templates that include this one.

Returns
The list of templates with an `"include"` dependency targeting this template.

get_imports() → List[*Dependency*]

Get all `"import"` dependencies.

Returns
The list of all `"import"` dependencies.

get_includes() → List[*Dependency*]

Get all `"include"` dependencies.

Returns
The list of al `"include"` dependencies.

get_parent() → Optional[*Dependency*]

Get the parent template, if there is one.

Returns
An `"extends"` dependency or `None`.

property name: str

The name of the template.

property was_modified: bool

True if the template was loaded multiple times.

class jinja2td.**Dependency**(*dependency_type: str*, *targets: List[*Target*]*, *with_context: Optional[bool] = None*, *ignore_missing: Optional[bool] = None*, *imported_as: Optional[str] = None*, *imported_names: Optional[List[str]] = None*)

A dependency to one or more templates.

property ignore_missing: Optional[bool]

Wether the dependency is optional or not.

Note: Only available with `"include"` dependecies.

property imported_as: Optional[str]

Wether the dependency is optional or not.

Note: Only available with `"import"` dependecies.

property imported_names: Optional[List[str]]

Wether the dependency is optional or not.

Note: Only available with `"import"` dependecies.

property resolved: List[str]

The names of the templates actually imported by this dependency.

Warning: Templates used by async environments aren't taken into account by default. See *DependencyGraph.used_last_watch* for more information.

property resolved_last_watch: List[str]

The names of the templates imported during the last watch.=

property target: Optional[*Target*]

The target of the dependency, if there is only one, or `None`.

Note: Use *targets* instead if you need to handle multiple targets.

property targets: List[*Target*]

All the targets of the dependency, in order.

For example, an `{% include ['template', 'fallback'] %}` will have two targets.

property type: str

The type of dependency. May be one of `"extends"`, `"include"` or `"import"`.

property with_context: Optional[bool]

Wether the context is passed to the dependency or not.

Note: Only available with `"include"` and `"import"` dependecies.

class jinja2td.**Target**(*dynamic: bool*, *name: Optional[str]*)

The target of a dependency.

A target is dynamic if it is not hard-coded in the template, in wich case the name of the target is unknown until it is resolved.

property is_dynamic: bool

True if the target name can't be known until the template is rendered.

property name: Optional[str]

The name of the target template, or *None* if the target is dynamic.

INDEX